MVE165/MMG631
Linear and Integer Optimization with Applications
Lecture 2
JuMP and Gurobi; Assignment 1

Edvin Åblad

2019–03–29

JuMP

- An open-source modelling language for mathematical optimization

  $\Longrightarrow$ Convenient and general interface to solvers
  $\Longrightarrow$ Formulate optimization models and examine solutions
  $\Longrightarrow$ Supports both open-source and commercial solvers

- A package in Julia

  $\Longrightarrow$ Efficient open-source script language
  $\Longrightarrow$ High level syntax similar to MATLAB
  $\Longrightarrow$ Python-like package manager

Gurobi

- Optimization *software package* for solving linear and quadratic optimization problems with continuous and integer variables
- Originally based on the *simplex method*, implemented in C
- The *primal and dual* simplex methods (see Lectures 3–5)
- The *barrier method*
- Techniques for avoiding *degeneracy* (see Lecture 4)
- Generating *cutting planes* (see Lecture 9)
- The *branch&bound* algorithm (see Lecture 8)
- *Heuristic* methods (see Lecture 10)

Using JuMP and Gurobi (computer exercise)

- Julia from `julialang.org`
- Use IDE (e.g., Juno) or the terminal and a text editor
- Type `]add JuMP#v0.18.5` in the Julia console
- Gurobi
  - Free for academic use
  - Set the path `GUROBI_HOME` (see exercise)
- Easy installation on private computer (see exercise)
  - *The teachers cannot, however, provide any technical support regarding these installations*

Solvers that work with JuMP

- **Gurobi** – solves linear and quadratic optimization problems with continuous and integer variables
- CPLEX – solves linear and quadratic optimization problems with continuous and integer variables
- MOSEK – solves linear and nonlinear optimization problems with continuous variables
- **Clp** – free, solves linear optimization problems with continuous variables
- **Cbc** – free, solves linear optimization problems with continuous and integer variables
- Baron, Ipopt, Knitro, Nlopt, Xpress, etc
- See JuMP documentation for table of supported solvers

The diet problem—description

- *The diet problem* (G.B. Dantzig, Interfaces 20(4):43–47, 1990) `https://resources.mpi-inf.mpg.de/departments/d1/teaching/ws14/Ideen-der-Informatik/Dantzig-Diet.pdf`
- Choose foods to meet certain nutritional requirements in the cheapest way
- A more sustainable version:
  - Kinds of food *[beans, egg, milk, potato, tomato]* are available in a limited amount per day and at a given price
  - 100g of each food provide given amounts of certain nutrients *[carbohydrates (CHO), protein, vitamin C, vitamin D]*
  - *Diet: requirements (upper and lower limits) on the daily amounts of each nutrient*

The Diet Problem—data

Food	price [SEK/hg]	available [hg/day]	CHO [g/hg]	protein [g/hg]	vit C [mg/hg]	vit D [μg/hg]
Beans	3.3	7	3.5	1.80	16.0	0
Egg	6.0	6	0.4	12.38	0	1.47
Milk	0.9	8	4.7	3.51	0.6	1.0
Potato	2.6	10	17.5	1.81	17.4	0
Tomato	5.8	5	2.6	0.81	14.8	0
Minimum amount/day			250 g	63 g	75 mg	10 μg
Maximum amount/day			300 g	125 g	1000 mg	1000 μg

* Data from `www.livsmedelsverket.se/livsmedelsdatabasen` and `www.coop.se/Handla-online/`

The Diet Problem—mathematical model

- Sets
  - $\mathcal{J} = \{1, \ldots, 5\}$ — kinds of food
  - $\mathcal{I} = \{1, \ldots, 4\}$ — nutrients
- Variables

The Diet Problem—mathematical model

- Sets
  - $\mathcal{J} = \{1, \ldots, 5\}$ — kinds of food
  - $\mathcal{I} = \{1, \ldots, 4\}$ — nutrients
- Variables
  - x_j, $j \in \mathcal{J}$ — purchased amount of food j per day [hg]
- Parameters

The Diet Problem—mathematical model

- Sets
  - $\mathcal{J} = \{1, \ldots, 5\}$ — kinds of food
  - $\mathcal{I} = \{1, \ldots, 4\}$ — nutrients
- Variables
  - x_j, $j \in \mathcal{J}$ — purchased amount of food j per day [hg]
- Parameters
  - c_j, $j \in \mathcal{J}$ — cost of food j [SEK/hg]
  - a_j, $j \in \mathcal{J}$ — available amount of food j [hg]
  - p_{ij}, $i \in \mathcal{I}$, $j \in \mathcal{J}$ — content of nutrient i in food j
    [g/hg], [g/hg], [mg/hg], [μg/hg]
  - n_i — lower limit on the amount of nutrient i per day
    [g], [g], [mg], [μg]
  - N_i — upper limit on the amount of nutrient i per day
    [g], [g], [mg], [μg]

The Diet Problem—mathematical model

The Diet Problem—mathematical model

$$\text{minimize} \quad \sum_{j \in \mathcal{J}} c_j x_j, \qquad \text{(good)}$$

The Diet Problem—mathematical model

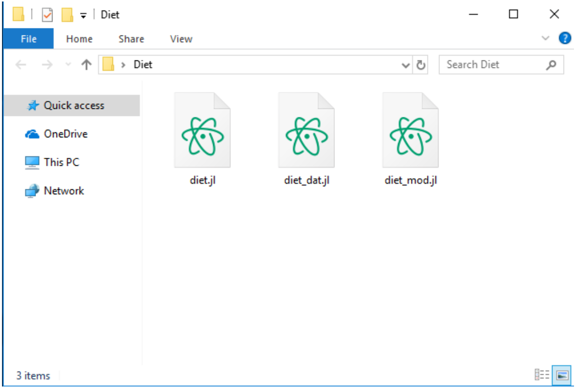
$$\text{minimize} \quad \sum_{j\in\mathcal{J}} c_j x_j, \qquad \text{(good)}$$

$$\text{subject to} \quad n_i \leq \sum_{j\in\mathcal{J}} p_{ij} x_j \leq N_i, \quad i \in \mathcal{I}, \qquad \text{(possible)}$$

$$0 \leq x_j \leq a_j, \quad j \in \mathcal{J}. \qquad \text{(possible)}$$

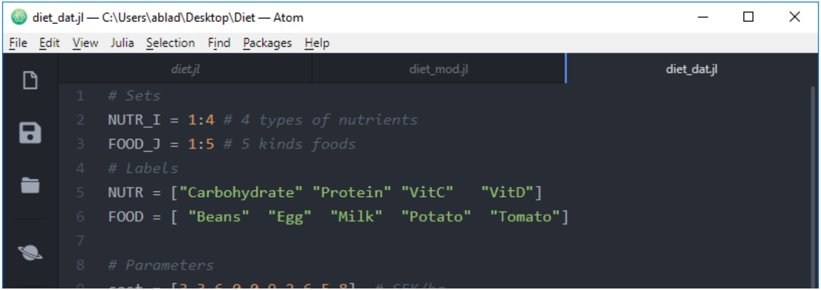
The Diet Problem—JuMP implementation

- Create a folder: *Diet*
- Create a file with the model: *diet_mod.jl*
- Create a data file: *diet_dat.jl*
- Create a main file: *diet.jl*



The Diet Problem—JuMP implementation

- Fill the data file using a text editor (Atom, Vim, Emacs, ...)
- Comments start with #
- Sets
  - $\mathcal{I} = \{1, 2, 3, 4\}$
  - $\mathcal{J} = \{1, 2, 3, 4, 5\}$



The Diet Problem—JuMP implementation

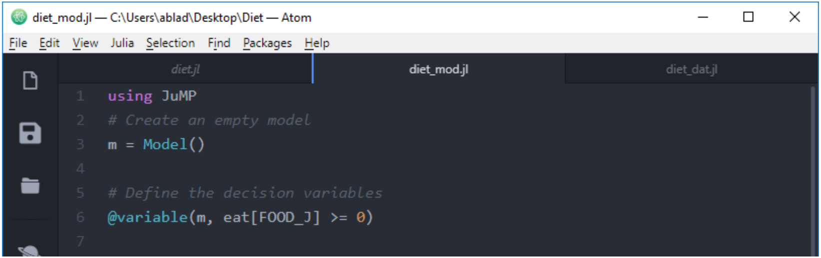
- Fill in the data file using the text editor
- Assign values to the parameters
- For large data sets use, e.g, `DelimitedFiles`

```

# Parameters
cost = [3.3 6.0 0.9 2.6 5.8]  # SEK/hg
eat_max = [7 6 8 10 5]        # hg/day

# nutrient requirements [g g mg µg]/day
n_min = [250 63 75 10]
n_max = [300 125 1000 1000]

# content of nutrient i in food j per 1hg food
#    Carb[g]   Prot[g]    VitC[mg]    VidD[µg]
content = [3.5     1.80     16.0     0         #Beans
           0.4    12.38      0       1.47      #Egg
           4.7     3.51      0.6     1.0       #Milk
          17.5     1.81     17.4     0         #Potato
           2.6     0.81     14.8     0]        #Tomato

```

diet_dat.jl 23:1 CRLF UTF-8 Spaces (4) Julia GitHub Git (0) Main

The Diet Problem—JuMP implementation

- Introduce variables: `@variable`
- Formulate non-negativity requirements
- Variables: x_j
  - $x_j \geq 0,\ j \in \{1, \ldots, 5\}$



The Diet Problem—JuMP implementation

- Formulate an objective function: `@objective`
- min $\sum_{j \in \mathcal{J}} c_j x_j$

```
# Define the decision variables
@variable(m, eat[FOOD_J] >= 0)

# Define the objective function
@objective(m, Min, sum(cost[j]*eat[j] for j in FOOD_J))

```

The Diet Problem—JuMP implementation

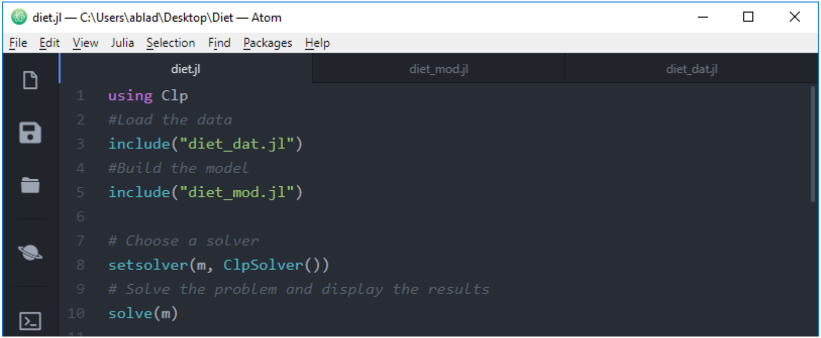
- Formulate constraints: `@constraint`
- Use arithmetic relations: `>=` , `<=` , `==`
- $n_i \leq \sum_{j \in \mathcal{J}} p_{ij} x_j \leq N_i,\ i \in \mathcal{I},$
- $x_j \leq a_j,\ j \in \mathcal{J}$

```julia

# Define the constraints on nutritional requirements
@constraint(m, nutr_cont[i in NUTR_I],
    n_min[i] <= sum(content[j, i]*eat[j] for j in FOOD_J) <= n_max[i])

# Define the constraints on the amount of food
@constraint(m, food_max[j in FOOD_J], eat[j] <= eat_max[j])

```

diet_mod.jl 17:1 CRLF UTF-8 Spaces (4) Julia GitHub Git (0) Main

The Diet Problem—JuMP implementation

- Fill the main file using the text editor
- Load the model and the data by including the files
- Choose solver: `setsolver`
- Solve the problem: `solve`



The Diet Problem—JuMP implementation

- Run the file and display results

```julia
# Solve the problem and display the results
solve(m)

eat_value = getvalue(eat)
tot_cost = getobjectivevalue(m)

for j in FOOD_J
    println("$(FOOD[j]) = $(eat_value[j]) hg/day")
end
println("Total cost = $tot_cost")
```

REPL

```
Beans = 7.0 hg/day
Egg = 1.3605442176870755 hg/day
Milk = 8.0 hg/day
Potato = 10.0 hg/day
Tomato = 4.7522239665096775 hg/day
Total cost = 92.02616431187857
julia>
```

diet.jl 23:4 CRLF UTF-8 Spaces (2) Julia GitHub Git (0) Main

The Diet Problem—JuMP implementation

- Define functions to structure your code
- To compute slack of a constraint:

```julia
# You can always define aid functions to simply your life, as below.
# Moreover, it's good practice to place these functions in a separate file
# and use include("name_of_that_file.jl"), to keep the code structured.
"""
 Gets the current slack of the constraint
"""
function getslack(constraint::ConstraintRef)::Float64
  lin_constr = LinearConstraint(constraint)
  row_val = getvalue(lin_constr.terms)
  return min(lin_constr.ub - row_val, row_val - lin_constr.lb)
end
```

- Julia is optionally typed, use `::Type` if you want to be precise

The Diet Problem—JuMP implementation

- Use `getdual` to
  - Get the reduced cost of a variable
  - Get the dual variable corresponding to a constraint

```julia
println("reduced costs =  ", getdual(eat))
println("dual variables =  ", getdual(nutr_cont))
println("slack of $(NUTR[2])  =  ", getslack(nutr_cont[2]))
println("slack of constraints =  ", getslack.(nutr_cont.innerArray))
```

REPL

```
reduced costs =  [0.0, 0.0, 0.0, 0.0, -8.88178e-16]
dual variables =  [2.23077, 0.0, 0.0, 3.47462]
slack of Protein  =  16.472838827838828
slack of constraints =  [0.0, 16.4728, 286.133, 0.0]
julia>
```

- In Julia `func.`(vec) applies `func` element-wise to vec
  $\Longrightarrow$ No need to define `getslack` for vectors of constraints

The Diet Problem—JuMP implementation

- Change type of variables: *integer*, *binary*, ...

```
# Define the decision variables
@variable(m, eat[FOOD_J] >= 0, Int)
```

- The sensitivity analysis as described is possible only for linear programs in continuous variables (not for integer/binary; this is due to theoretical properties (see the course book, Ch. 5))

The Diet Problem—JuMP implementation

- Solution to the integrality constrained model

```julia
# Choose a solver
using Gurobi
setsolver(m, GurobiSolver())
# Solve the problem and display the results
solve(m)
```

REPL

```
Beans = 7.0 hg/day
Egg = 2.0 hg/day
Milk = 8.0 hg/day
Potato = 10.0 hg/day
Tomato = 5.0 hg/day
Total cost = 97.3
julia>
```

- Theory for linear optimization problems with integer/discrete constraints: see the course book and Lectures 7–10

The Diet Problem—JuMP implementation

- Read and write to file

```julia
#Read and write to files (column separated data)
using DelimitedFiles
A = rand(3,5)
writedlm("output.txt", A, '\t')
B = readdlm("output.txt", '\t', Any, '\n')
```

```
julia> all(B .== A)
true
```

- Read large datasets
- Export result

Global and local variables in Julia

- `global` ⇒ read everywhere, can't modify in loops or functions
- `local` ⇒ read and modify, only exist in the current block

```julia
x = 0 # A global variable.
for i = 1:10
    y = i   # y is declared inside a loop, it is thus local.
    # You can read the value of x.
    println("x = ", x)
    # x = y, NOT OK writing to a global variable inside a loop.
end
# y do not exist here.
```

Global and local variables in Julia

- In functions variables are declared local

```
function do_something()
    z = 0 # z is declared in a function, and is thus local.
    for i = 1:10
        z += 1 # ok since z is local
    end
    return (z + x)
end
x = 2
println("z + x = ", do_something()) # prints: z + x = 12
```

- Functions also keeps the code structured

Misc Julia info

- Package manager: `]add pkgname`
- Information on obj: `?obj`
- Dictonary, possible to create named data

```
costs = Dict{String,Float64}()
costs["Beans"] = 3.3
costs["Egg"] = 6.0
```

- Solver options:

  `setsolver(m, ClpSolver(...))`
- `Plots` is a package for plotting in Julia
- Documentations
  - Julia: `https://docs.julialang.org/en/v1/`
  - JuMP: `http://www.juliaopt.org/JuMP.jl/v0.18/`
- We use version v0.18.5 not v0.19 of JuMP

Assignment 1: Biofuel supply chain

Chalmers University of Technology
University of Gothenburg
Mathematical Sciences
Optimization
Zuzana Nedělková
Ann-Brith Strömberg
Caroline Granfeldt

MVE165
MMG631
Linear and integer optimization
with applications

Assignment information
March 26, 2019

Assignment 1: Biodiesel supply chain

The questions 1, 2, and 3a–3f are mandatory. Students aiming at grade 4, 5, or VG must also answer the questions 3g–3i, and the quality of the assignment report must be high.

Problem background

The background of this problem comes from a study of the biofuels supply chain in Greece performed during the years 2006–2010 by Papapostolou et al. and reported in [4]. The numerical data are collected from [1, 3, 2], then slightly modified.

The following sections present a short description of biodiesel production. After this follows a short description of the plants that can be used and of the final products with their demands. The description includes prices, yields, availabilities of different sources, demands, and data regarding the different production processes.

1. Biofuels supply chain

The biodiesel is produced in a two-step process, first the extraction of vegetable oil from seeds, after this the transesterification in order to produce the pure biodiesel.

In the extraction phase the vegetable oils is gained from the seeds through a simple screw press. The vegetable oil content in the respective kind of seeds is presented in Table 1.

All extracted vegetable oil is transesterificated to produce biodiesel. The transesterification is a process in which the vegetable oil is being reacted with methanol. The biodiesel is the product resulting from this reaction. The produced biodiesel has to be refined to remove impurities. To produce 0.9 l of biodiesel we need 1 l of vegetable oil and 0.2 l of methanol. The price of methanol is 1.5 €/l.

1

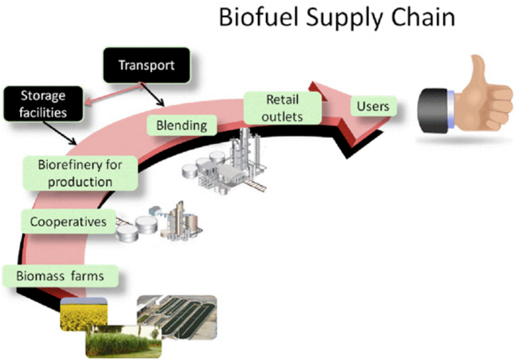
Biofuel supply chain

- Reduce oil dependence
- Reduce greenhouse effect and climate change
- Substitute fuel in transportation sector
- Biofuels can be used in existing cars
- EU quotas to use
  - 10% of energy in transport from renewable sources by 2020
  - 5% of biodiesel in diesel fuel from 2003
- Food versus fuel debate ...
- Develop a mathematical model of the biofuel supply chain

Biofuel supply chain

The value chain typically includes:

- Feedstock production
- Biofuel production
- Blending
- Distribution
- Consumption



Assignment 1: Biodiesel supply chain

- Biodiesel supply chain problem
  - Maximize the total profit
  - Supply the demand of biodiesel
- Tasks
  - Formulate a linear optimization model
  - Model and solve the problem using JuMP and Gurobi (or another LP-solver)
  - Perform sensitivity analyses

Crops

- Data
  - Available area for growing crops
  - Crops: Soy, Sunflower, Cotton
  - Each crop yields an expected amount of seeds
  - Each crop has a water demand
  - The available water is limited
- Processes
  - Extraction of vegetable oils from seeds (given yields)
  - Transesterification: vegetable oil + methanol = biodiesel (given proportions)
  - Purchase methanol (given price)

Final Products

- Data
  - Three different products/blends: B5, B30, B100
  - Each product has price
  - Each product is subject to tax (higher amount of biodiesel ⇒ lower tax)
  - Demand of fuels to be delivered
- Processes
  - Blending of biodiesel and petrol diesel
  - Purchase petrol diesel (given price and availability)

Sensitivity analysis

- Analyze results and answer several important questions
- How sensitive is the optimal solution and the optimal value to changes in the data? (Course book ch. 4–6 & lectures 4–6)
  - *Reduced costs* of a non-basic variable: the change in the objective value when the value of the corresponding variable is (marginally) increased
  - *Shadow price* of a constraint: the change in the optimal value when the RHS is (marginally) changed; equals the optimal value of the corresponding *dual variable*
  - The optimal value of the *slack variable* of a constraint indicates how much the RHS can be reduced while staying feasible
- Use these concepts to answer the questions

Others

- Cetane number
  - The quality of pure biodiesel is given by the cetane number
  - The cetane number depends on the quality of the crops
  - Requirements for the quality of each product should be incorporated in the model
- Environmental friendly objective function

Literature

- I. Dunning and J. Huchette and M. Lubin, *JuMP: A Modeling Language for Mathematical Optimization*, SIAM Review, 2017, http://www.juliaopt.org/JuMP.jl/v0.18/

- Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2018, http://www.gurobi.com/documentation/8.1/refman/

- Z. Nedělková, A.-B. Strömberg, C. Granfeldt, *Assignment 1: Biodiesel supply chain*, March 26, 2019, http://www.math.chalmers.se/Math/Grundutb/CTH/mve165/1819/#Assignments

- C. Papapostolou, E. Kondili, J.K. Kaldellis, *Development and implementation of an optimisation model for biofuels supply chain*, Energy, Volume 36, Issue 10, October 2011, Pages 6019–6026

- J. Lundgren, M. Rönnqvist, P. Värbrand, *Optimization*, Studentlitteratur AB, Lund, 2010